Migrating to Java 25 | Java 25 New Features and Skills - TT2137

Fast-paced, hands-on course designed for developers with prior experience in earlier versions of Java who need to quickly get up to speed with the latest innovations.

Duration: 1 Day
Skill Level: Introductory
Available Format: Instructor-Led Online; Instructor-Led, Onsite In Person ; On Public Schedule

What You'll Learn

Overview

There have been significant advancements in the Java programming language since the release of Java 8 in March of 2014. This one-day course focuses on the features and improvements that are most impactful for developers who are transitioning to Java 25, including enhancements to the language such as records, sealed classes, switch expressions, pattern matching, flexible constructors, module import declarations, and scoped values.

This is a fast-paced, hands-on course designed for developers with prior experience in earlier versions of Java who need to quickly get up to speed with the latest innovations. The course emphasizes not only how to use these new features, but also best practices for adopting them in real-world projects to write cleaner, more maintainable, and more efficient code.

This is not just a class about the differences between Java 21 and Java 25. To give participants the full context, the course also reviews key enhancements introduced since Java 8, including the Java Module System, text blocks, records, and sealed classes. Many of these earlier innovations laid the groundwork for the modern features now solidified in Java 25, and understanding their evolution helps developers appreciate the design choices in the current release.

By the end of the course, participants will be equipped with the skills to confidently modernize their applications, make the most of new Java 25 capabilities, and ensure smooth migration paths from legacy Java versions to the latest platform.

Objectives

This *skills-centric* course is approximately 50% hands-on lab and 50% lecture, designed to train attendees in advanced Java development skills by combining the most current techniques with proven industry practices. Working in an engaging, hands-on learning environment guided by our expert instructors, attendees will learn to:

- Distinguish between **LTS (Long-Term Support)** and **non-LTS** Java versions.
- Evaluate the advantages and disadvantages of using **preview features** in production environments.
- Implement data objects using the **record type** to simplify immutable class creation.
- Improve memory consumption and efficiency through the use of **compact strings**.
- Apply the new **strip()** and **isBlank()** methods in the String class to handle whitespace effectively.
- Define and use **multi-line text blocks** for cleaner string formatting.
- Use **switch expressions** to assign values and simplify branching logic.
- Understand the concept of **pattern matching** and how it improves code readability.
- Apply **pattern matching for instanceof** to streamline type checking.
- Use **pattern matching in switch statements** to simplify conditional logic.
- Design applications using **sealed classes and interfaces** to enforce controlled inheritance hierarchies.
- Review and apply the **Java modular system (Project Jigsaw)** for scalable application design.
- Explore **virtual threads** to simplify concurrent programming and improve application scalability.

Specific Java 25 features that are covered in the course include:

- Utilize **scoped values** as a safer, lightweight alternative to ThreadLocal for passing data across threads.
- Take advantage of **module import declarations** to make modular code more readable and maintainable.
- Implement **flexible constructor bodies** to gain more control over object initialization.
- Experiment with **compact source files and instance main methods** to reduce boilerplate and simplify beginner-level or quick-start applications.

Audience

This course is designed for professional Java developers who are already proficient in the language and want to update their skills to take advantage of the features introduced through Java 25. It is particularly well-suited for developers who have been working with Java 8 or later and are now preparing to modernize their applications or development practices.

The class is ideal for individuals responsible for maintaining legacy systems, transitioning projects to newer Java releases, or adopting modern Java techniques to improve code clarity, scalability, and performance. Team leads, senior developers, and architects will also benefit from understanding how newer Java features fit into enterprise development and long-term platform strategy.

Pre-Requisites

This course is intended for experienced Java developers who already have a solid working knowledge of the language. Because this course builds on the evolution of the platform, participants are expected to be familiar with Java 8 features and constructs, including lambda expressions, the Stream API and functional interfaces. Prior experience with modern IDEs and build tools is also assumed.

This is not a beginner's course; it is designed for developers who are already productive in Java and want to upgrade their skills to take advantage of new features through Java 25.

Agenda

1) Versions and Features

This lesson reviews the evolution of Java since version 8, highlighting important API updates and providing context for why modern features were introduced. Provide an overview of the major Java versions released since Java 8.

- Explore the most important API updates introduced since Java 8.
- Tutorial: Importing code Using Maven

2) Review of the Java Module System

The Java Module System remains a cornerstone for scalable application design. In this lesson, students revisit modularity concepts and learn how service discovery is supported through the ServiceLoader mechanism.

- Review the purpose and structure of the Java Module System.
- Define and configure modules for scalable application design.
- Work with the ServiceLoader to explore modular service discovery.
- **Lab:** Defining Modules
- **Lab:** Modules and the ServiceLoader (optional)

3) Cleaner Code in Java 25

Java 25 introduces features that streamline code structure and reduce boilerplate. This lesson covers module import declarations, flexible constructor bodies, and compact source files with instance main methods, showing how these updates simplify development

- Use module import declarations to simplify module dependencies.
- Apply flexible constructor bodies to improve clarity and flexibility in object initialization.
- Write simple programs using compact source files and instance main methods to reduce boilerplate.
- Lab: Compact Source files

4) Records

Records provide a concise way to define immutable data carriers. This lesson introduces records, demonstrates their constructors, and explains how they integrate with other modern Java features.

- Explain how records are used as lightweight, immutable data objects in Java.
- Define records to simplify class design.
- Implement both canonical and compact constructors for records.
- **Lab:** Records

5) String and Text Blocks

Working with text is easier and more efficient in modern Java. This lesson explores string enhancements such as strip(), isBlank(), repeat(), compact strings, and multi-line text blocks with formatting and indentation support.

- Discuss how whitespace is defined and handled in Java.
- Use the strip() methods of the String class to manage whitespace.
- Improve memory efficiency through the use of compact strings.
- Define and use text blocks for clean, multi-line string literals.
- Apply formatting and indentation techniques when working with text blocks.
- **Lab:** Text Blocks

6) Sealed Classes

Sealed classes and interfaces provide fine-grained control over inheritance. This lesson explains how sealed types work, how to restrict subclassing, and how they improve code clarity when combined with pattern matching.

- Introduce sealed classes as a way to control inheritance hierarchies.
- Use the sealed and permits modifiers to restrict subclassing.
- Define and apply sealed interfaces.
- **Lab:** Sealed Classes

7) Switch Expressions

Switch expressions modernize one of Java's oldest constructs. In this lesson, students learn how switch expressions simplify conditional logic, how to use the yield keyword, and how to handle fall-through more clearly.

- Use switch expressions to simplify conditional logic.
- Apply the yield keyword for returning values from switch branches.
- Understand and manage switch fall-through in modern switch constructs.
- **Lab:** Switch Expressions

8) Pattern Matching

Pattern matching has matured into one of the most powerful tools in modern Java. This lesson covers instanceof pattern matching, switch patterns and, record patterns.

- Apply pattern matching for instanceof to simplify type checks.
- Use pattern matching in switch statements for cleaner conditional logic.
- Combine pattern matching with sealed classes for safer and more concise code.

- Explore record patterns for deconstructing record components.
- Combine sealed classes with pattern matching for more expressive code.
- **Lab:** Pattern Matching

9) More Updates

This lesson highlights additional modern Java features that developers need to know, including sequenced collections, private methods in interfaces, enhancements to the @Deprecated annotation, multi-release JAR files, and Javadoc improvements.

- Explore sequenced collections and their role in ordered data handling.
- Define and use private methods in interfaces.
- Apply the forRemoval and since attributes of the @Deprecated annotation.
- Understand the purpose and benefits of multi-release JAR files.
- Review and apply important updates to Javadoc.
- **Lab:** Creating a Multi-Release Jar file (optional)

10) Virtual Threads

Concurrency in Java has been transformed by virtual threads. This lesson introduces virtual threads, compares them to platform threads, and explores scoped values as a safer alternative to ThreadLocal.

- Explain the purpose and advantages of using virtual threads.
- Differentiate between virtual threads and traditional platform threads.
- Create and manage virtual threads in Java applications.
- Use scoped values as a safer alternative to ThreadLocal for sharing state in concurrent programming.
- **Lab:** Virtual Threads

This course can be taught using a **local installation model**, where students install the Java Development Kit (JDK 25) and IntelliJ IDEA on their own systems. This approach allows participants to work directly on their personal development environment, gaining familiarity with tools they are likely to use in real-world projects. Installation instructions and guidance are provided before the course to ensure everyone is prepared to begin hands-on exercises right away.

Alternatively, the course can be delivered using our **Remote Desktop solution**, which provides students with access to a fully pre-configured environment. This setup includes JDK 25, IntelliJ IDEA, and all supporting tools installed and ready to use. The remote desktop option eliminates the need for local configuration and ensures a consistent, reliable experience for all participants, accessible through a standard Remote Desktop client or web browser. **This approach also enables the instructor to access each student's environment directly when needed, with full keyboard and mouse control, without ever accessing the student's personal computer. In addition, it reduces the risk of accidental sharing of company or personal information during screen sharing, since all work takes place within the controlled remote environment.**

For More Information

Please contact us or call 844-475-4559 toll free for more information about our training services (instructor-led, self-paced or blended), coaching and mentoring services, public course enrollment or questions, partner programs, courseware licensing options and more.